ITE5315 Project

Project duration: Week12~14

Project submission:

- Phase 0: Apr 3 during the class, project planning, basic setup
- Phase 1:, Apr 9 submission + Apr 10 class discussion (attendance is mandatory)
- Phase 2: Apr 16 (Complete/Submit project + video)
- Phase 3: Apr 17Project presentation/evaluation

Project Team: Team of two

Assessment Weight: 30% of your final course Grade

Assessment includes:

- Phase 0(2%), Phase 1(8%), Phase1-discussion(3%), Phase 2(12%), Project peer evaluation + presentation in Week 13 (5%)

Submission Deadline: Before Week14 class

Objective: To explore more on developing db-driven Node/Express app (REST API)

Description: Using the concepts that we learned in the course and, we are going to develop a secure db-driven Node/Express app.

Project Specification:

Step 1: Loading the "Sample Movie(mflix) Data" in MongoDB Atlas

Step 2: Building a Web API

- You need to install express, cors, mongoose + set up Git repository
- Add a module to interact with Movie MongoDB (Similar to assignment 3)
    - "Initializing" the Module before the server starts
        - To ensure that we can indeed connect to the MongoDB Atlas cluster with our new connection string, we must invoke the db.initialize("connection string…") method and only start the server once it has succeeded, otherwise we should show the error message in the console

- This module will provide the 6 (promise-based) functions required by our Web API for this particular dataset
- db.initialize("Your MongoDB Connection String Goes Here"): Establish a connection with the MongoDB server and initialize the "Movie" model with the "Movie" collection (used above)
- db.addNewMovie(data): Create a new Movie in the collection using the object passed in the "data" parameter
- db.getAllMovies(page, perPage, title): Return an array of all Movies for a specific page (sorted by Movie_id), given the number of items per page. For example, if page is 2 and perPage is 5, then this function would return a sorted list of Movies (by Movie_id), containing items 6 – 10. This will help us to deal with the large amount of data in this dataset and make paging easier to implement in the UI later. Additionally, there is an optional parameter "title" that can be used to filter results by a specific "title" value
- db.getMovieById(Id): Return a single Movie object whose "_id" value matches the "Id" parameter
- updateMovieById(data,Id): Overwrite an existing Movie whose "_id" value matches the "Id" parameter, using the object passed in the "data" parameter.
- deleteMovieById(Id): Delete an existing Movie whose "_id" value matches the "Id" parameter

Add the routes : The next piece that needs to be completed before we have a functioning Web API is to actually define the routes (listed Below). Note: Do not forget to return an error message if there was a problem and make use of the status codes 201, 204 and 500 where applicable.

- POST /api/Movies
  - This route uses the body of the request to add a new "Movie" document to the collection and return the created object / fail message to the client.
- GET /api/Movies
  - This route must accept the numeric query parameters "page" and "perPage" as well as the (optional) string parameter "title", ie: /api/movies?page=1&perPage=5&title=The Avengers. It will use these values to return all "Movie" objects for a specific "page" to the client as well as optionally filtering by "title", if provided (in this case, it will show both “The Avengers” films).
  - EXTRA CHALLENGE (bonus): add query param validation to your route in order to make sure that the params you expect are present, and of the type you expect. You can do this using packages like https://www.npmjs.com/package/celebrate or https://express-validator.github.io/docs/check-api.html . If the params are incorrect, your route should return a 400 response (client error) vs. 500 (server error).
- GET /api/Movies
  - This route must accept a route parameter that represents the _id of the desired movie object, ie: /api/movies/573a1391f29313caabcd956e. It will use this parameter to return a specific "Movie" object to the client.

- PUT /api/Movies
    - This route must accept a route parameter that represents the _id of the desired movie object, ie: /api/movies/573a1391f29313caabcd956e. It will use this parameter to return a specific "Movie" object to the client.

- DELETE /api/Movies
    - This route must accept a route parameter that represents the _id of the desired movie object, ie: /api/movies/573a1391f29313caabcd956e as well as read the contents of the request body. It will use these values to update a specific "Movie" document in the collection and return a success / fail message to the client.

Step 3: Add UI/Form

You want to demonstrate your skill in working with Template Engines and Form, but you don't want to apply this for the entire application.
Try to make a new route which works similar to "/api/Movies?page=1&perPage=5&title= The Avengers" and take the 'page', 'perPage' and 'title' through FORM and display the output using Template Engine!

Use your creativity to design the layout and apply proper css style/format.

Step 4 : Add security feature to the app:

A) **Use Environment Variable for your Connection String:** Your solution currently has your database connection string (with username and password!) hard coded into your source code. This is a potential security risk. If this code is shared between users on a team, or if it is pushed to a public repo on GitHub, your password is now public too.

B) The app gives access to Movie data. How do you limit the user-access, so that only authorized users can open the app or access to specific route?

a. You are asked to use the security features like Password Encryption, JWT, Session, Cookie that we have learned in the course in order to allow only authorized user to use some of the special routes! You may also think about adding "API-key" to the route (like "openweathermap") for the authorized user

b. The goal is encrypt sensitive data in DB and use JWT(full mark) to secure routes for authorized users (users can be designed locally in app or from a DB collection). However, you can plan other simpler implementations and still get full mark depends to your strategy.

Step 5 : Add new functionality/feature to the app

You can use your creativity to add a new functionality to the app. This can be adding a new UI/route/DB operation, or using a new npm package that enhances your design.

Step 6: Pushing to Cyclic

Once you are satisfied with your application, deploy it to Cyclic

Bonus :

- **(additional 5%):** Add the GraphQL to the app in order to extract and manipulate Movie data. This can be applied into the data extraction or even data manipulation routes.
- **(additional 2%):** Design a JS front-end (jQuery , Native JS, React, …) to utilize the API that you designed and use your creativity to add some bootstrap flavor to that ☺

Project expectation:

- Following best practices in handling async tasks
- Following best practices in error handling
- Practice good way of structuring NodeJs app and separate functionalities/layers
- Practice good way of commenting codes
- I am open to utilizing new techniques/packages as far as you are able to understand and explain it. However, I would prefer to see you use the practices that we did in other activities during this course.

Project Submission:

- Add the following declaration at the top of .js files

```
/*********************************************************************
***
* ITE5315 – Project
* I declare that this assignment is my own work in accordance with Humber Academic Policy.
* No part of this assignment has been copied manually or electronically from any other source
* (including web sites) or distributed to other students.
*
* Group member Name: _____ Student IDs: _____ Date: _____
*********************************************************************
***/
```

- Compress (.zip) the files in your Visual Studio working directory (this is the folder that you opened in Visual Studio to create your client side code).
- Complete & Submit the project document (given template).
- Record a detailed walk-through/demonstration video presentation of the project

Important Note:

- Submitted assignments must run locally, ie: start up errors causing the assignment/app to fail on startup will result in a **grade of zero (0)** for the assignment.
- **LATE SUBMISSIONS for assignments**. There is a deduction of 1% per hour for Late submissions, and after two days it will grade of zero (0).
- Assignments should be submitted along with a video-recording which contains a detailed walkthrough of solution. Without recording, the assignment can get the maximum of 1/3 of the total.